ACEEE

Proc. of Int. Conf. on Advances in Computer Science, AETACS

Distributed Algorithm for Frequent Pattern Mining using HadoopMap Reduce Framework

Suhasini A. Itkar[1], Uday V. Kulkarni[2]
[1] PES Modern college of Engineering, Pune, India
Email: Suhasini.a.itkar@gmail.com
[2]SGGS Institute of Engineering and Technology, Nanded, India
Email: kulkarniuv@yahoo.com

***Abstract*—With the rapid growth of information technology and in many business applications, mining frequent patterns and finding associations among them requires handling large and distributed databases. As FP-tree considered being the best compact data structure to hold the data patterns in memory there has been efforts to make it parallel and distributed to handle large databases. However, it incurs lot of communication over head during the mining. In this paper parallel and distributed frequent pattern mining algorithm using Hadoop Map Reduce framework is proposed, which shows best performance results for large databases. Proposed algorithm partitions the database in such a way that, it works independently at each local node and locally generates the frequent patterns by sharing the global frequent pattern header table. These local frequent patterns are merged at final stage. This reduces the complete communication overhead during structure construction as well as during pattern mining. The item set count is also taken into consideration reducing processor idle time. Hadoop Map Reduce framework is used effectively in all the steps of the algorithm. Experiments are carried out on a PC cluster with 5 computing nodes which shows execution time efficiency as compared to other algorithms. The experimental result shows that proposed algorithm efficiently handles the scalability for very large datab ases.**

***Index Terms*—Distributed computing, Parallel processing, Hadoop Map Reduce framework, Frequent pattern mining, Association rules, Data mining**

I. INTRODUCTION

In data mining, research in the field of distributed and parallel mining is gaining popularity as there is tremendous increase in database sizes stored in data warehouses. Frequent Pattern Mining is one of the important areas in Data Mining to find associations in databases.

Apriori [1] algorithm is first efficient algorithm to find frequent patterns. It is based on candidate set generate and test approach. It requires n database scans for finding n itemsets in worst case scenario. Many parallel and distributed algorithms are developed using Apriori principle to improve efficiency of Apriori algorithm. Agarwal, Schafer [2] proposed in 1996 algorithms like count distribution, data distribution and candidate distribution based on parallel and distributed processing. In this paper communication, computation efficiency, memory usage and synchronization parameters are discussed. Zaki [3] in 1999 discussed about various aspects of parallel and distributed association rule mining algorithm. Other parallel and distributed algorithms also tried to improve performance based on Apriori principle like Trie tree [Bodon, 2003] based parallel and distributed algorithm [4], DPA (Distributed ParallelApriori) Algorithm [5], MapReduce

DOI: 02.AETACS.2013.4.123

programming model based Apriori algorithm using Hadoop framework [6], parallel data mining on shared memory system[14] and tree projection algorithm[15]. All these algorithms still have a performance bottleneck of either multiple database scans or huge search space or both.

In 2000 Han et al. came up with FP Growth [7] algorithm where it proposed more compact tree structure called FP-tree. It overcomes drawback of Apriori algorithm by reducing search space and taking only two database scans. It uses FP-growth algorithm for mining frequent patterns.

FP-growth suffers performance issues with huge databases. To improve performance of FP-tree based algorithms new area of research is introduced with distributed and parallel processing as discussed in survey paper of frequent pattern mining [16]. Proposed algorithms in this areatried to reduce IPC cost, memory storage and tried to use computing power

of processors efficiently with load balancing approach. Some of FP-tree based distributed algorithms are PP tree [8], parallel TID based FPM [9], distributed DH-TRIE [10] frequent pattern mining algorithm and efficient frequent pattern mining algorithms in many task computing environment[17].

As there is huge demand in data mining to process terabyte or petabyte of information spread across the internet, there was a need of more powerful framework for distributed computing. With the emergence of cloud computing environment,the Map-reduce framework patented by Google also gained popularity because of its ease of parallel processing, capability of distributing data and load balancing. Hadoop is an open source implementation of Map-reduce framework.

This paper proposes distributed and parallel frequent pattern mining algorithm (DPFPM) using HadoopMapReduce framework. Here proposed algorithm tries to reduce communication cost among all processors, and to speed up mining process. The proposed algorithm efficiently handles the scalability for very large databases. It shows best performance results for large databases using MapReduce framework on Hadoop cluster.

The remainder of this paper is organized as follows: Section II reviews FP-tree based parallel and distributed algorithms. Section III describes HadoopMapReduce framework. Section IV introduces a novel distributed and parallel frequent pattern mining algorithm (DPFPM) using HadoopMapReduce framework. Section V shows experimental results and comparison of proposed algorithm with existing approaches. Finally, Section VI draws conclusion from this study and discuss about future scope.

II. RELATED WORK

Frequent pattern mining problem is to find all frequent k-itemsets where all itemset support should be more than minimum support thresholdξ.

Support(k) >= min_sup_threshold$\xi$for given threshold ($1 \leq \xi \leq |DB|$).

Apriori based algorithms tried to solve the FPM problem but faced a performance bottleneck of either multiple database scan or huge search space or both. Prefix tree based algorithms like FP-growth tried to optimize performance but suffer performance issues with huge database and small support values where it takes more time for mining or it fails to execute. In this section previous work in the field of distributed and parallel frequent pattern mining is covered. Distributed algorithms based on prefix tree structure and MapReduce framework is discussed.

In PP tree based parallel and distributed algorithm[8], it proposes parallel and distributed algorithm using PP tree (parallel pattern tree) where it requires only one database scan to construct PP tree and in a way reduces I/O cost. It considers a distributed memory architecture where each node contains all resources locally.Algorithm is divided in three phases – Phase I: It first accept database contents in horizontal partitions in any canonical order, construct tree in one scan and then restructure it in global frequency descending sequence of items. Phase II: Local Mining – Individually mine local PP tree in parallel for discovering global frequent patterns (FP_{pg}). Phase III: Final sequential step which collects frequent patterns from all local PP trees and generate global frequent patterns. Phase I: PP tree construction phase – This phase first partitions original database and distribute it to all nodes. It then performs three steps at each local node. Step I: It construct local PP tree with insertion stage by generating local header table at each node. Step II: All local header tables are collected in global header table array (HTA).This global HTA is reorganized in frequency descending order. Step III: At each node restructuring phase is carried out with the help of global HTA and finally local PP tree is created.Phase II: Mining of local PP tree – PP tree generate the set of potential frequent patterns (FP_{pg}) by using FP Growth mining algorithm. For this first local PP trees construct itemset based prefix pattern trees $PT_{(x)}$ for each frequent pattern x. It is constructed by collecting all prefix sub paths/patterns from original local PP tree. From this individual tree for each frequent pattern x it generates new conditional tree $CT_{(x)}$, which

containsonly potential global items, it prunes all infrequent items. Then it generates the set of frequent patterns (FP_{pg}) at each local node in parallel without any IPC cost.Phase III: Generating global frequent patterns – In this master node takes local frequent patterns (FP_{pg}) and sequential step of accumulating supports of all local (FP_{pg}) is carried out at master node. The final (FP_{pg}) list may lead to false positive patterns but not false negatives. Master node then removes all infrequent patterns from (FP_{pg}) list and generates final global frequent patterns. For removing infrequent patterns from (FP_{pg}) list algorithm uses hash based technique to speed up the search in (FP_{pg}) list.

Although this algorithm tries to reduce Inter Process Communication cost with efficient mining algorithm and also reduces I/O cost by performing single database scan still it requires more time for insertion and restructuring phase while constructing local PP trees. Original database is partitioned but not pruned with infrequent 1-itemsets and original database is only used throughout the algorithm so this may utilizes more memory for huge databases.

In TPFP tree and BTP tree based parallel and distributed algorithm [9], two parallel and distributed mining algorithms are proposed based on Tidset concept. Tidset based parallel FP tree (TPFP tree) algorithm is proposed for cluster environment and balanced Tidset based parallel FP tree (BTP tree) algorithm is proposed for grid environment.In both the algorithms Tidset (Transaction identifier set) is used to directly select transactions instead of scanning whole database. Its goal is to reduce IPC cost and tree insertion cost, thus decreasing execution time. TPFP tree based algorithm is proposed for homogeneous cluster environment. In this algorithm load is distributed to all nodes evenly without viewing processor capacity. Its objective is to reduce the execution time of mining information exchange and to shorten the index cost of transaction extraction. BTP tree algorithm is proposed for heterogeneous grid environment where it tries to balance the load. In BTP algorithm all the steps are same as TPFP algorithm only one more step is added to it, where it evaluates the performance index of computing nodes. Depending upon performance index it distributes mining sets. This algorithm performs better than PFP tree and TPFP tree in grid environment.

In MapReduce as a programming model for Association Rules algorithm on Hadoop [6] paper, Apriori algorithm is improved using MapReduce model to handle huge datasets on Hadoop distributed computing environment. In this paper, it first finds frequent 1-itemsets in first database scan. Then it uses MapReduce model where each mapper generate subset of candidate itemsets. These candidate itemsets are given to reducer to prune candidates based on minimum support threshold and to generate subset of frequent itemsets. Multiple iterations to MapReduce are necessary to produce frequent itemsets. At final stage output of all reducers is combined to generate final frequent itemsets.In this paper although it tried to implement Parallel Apriori algorithm but it suffers from drawback of multiple database scans and huge candidate set generation.

A distributed DH-TRIE frequent pattern mining algorithm [10] is proposed using HadoopMapReduce programming model. This paper is based on Dynamic Hash (DH) Trie algorithm. Three problems are focused in this are globalization, random-write and duration which are tried to solve using Java Persistent API(JPA) and parallelizing algorithm using MapReduce programming model.In first phase, first two database scans are assigned to map tasks which results into building of DH-TRIE. In second phase mining is carried out using breadth-first search model by using queue instead of recursive processing for mining. This is carried out by multiple mapper tasks where number of mappers is equal to number of elements in the queue.Although this algorithm tries to solve three problems of globalization, random-write and duration it still faces some drawbacks like in first phase two database scans are handled by only map tasks, no reducer is used to construct FP-tree. In second phase mining is also carried out by map tasks to handle sub FP-tree, slowdowns mining process.

Therefore, this paper proposes DPFPM algorithm to speed up the process of frequent pattern mining for huge datasets by distributing the data efficiently and executing task in parallel. Proposed approach tries to overcome the drawbacks of previous work and accelerate the performance effectively.

III. HADOOPMAP-REDUCE FRAMEWORK

Hadoop is an open source programming framework for running applications on large clusters built of commodity hardware [11]. Hadoop is derived from Google's MapReduce model and Google File System. Hadoop cluster characteristic is partitioning or distributing data and computation across multiple nodes and executing computations in parallel. It provides application both reliability and data motion.It provides a Hadoop distributed file system (HDFS) [12] that stores data on the compute nodes, providing very high aggregate bandwidth across the cluster.

Hadoop implements a computational paradigm named MapReduce [13], where the application is divided into many small fragments of work, each of which may be executed or re executed on any node in the cluster. Its goals are to process large data sets, high throughput, load balancing and failure handling. MapReduce framework executes user defined MapReduce jobs across the nodes in the cluster.

A MapReduce computation has two phases map phase and reduce phase. The input to the computation is a data set of key/value pairs. It partition input key/value pairs into chunks, run map() tasks in parallel. After all map()s are complete, it consolidate all emitted values for each unique emitted key.

$$Map(K_1, V_1) \rightarrow list(K_2, V_2)$$

It partition space of output map keys, and run reduce() in parallel. Reducer generates output and stores it in HDFS.

$$Reduce(K_2, list(V_2) \rightarrow list(V_3)$$

Tasks in each phase are executed in a fault-tolerant manner. Having many map and reduce tasks enables good load balancing. It provides an interface which is independent of backend technology.

IV. DISTRIBUTEDANDPARALLELFREQUENTPATTERNMININGALGORITHM(DPFPM)

Proposed DPFPM algorithm uses HadoopMapReduce programming framework for frequent pattern mining. In frequent pattern mining the task of generating conditional frequent patterns and finding global frequent patterns can be done in parallel by distributing the data so MapReduce framework is most suitable for it.

In proposed algorithm MapReduce task is used twice, once for support counting and then for generating frequent patterns, takingmorebenefits of MapReduce framework. It achieves distributed system goals like high avaiability, high throughput, load balancing and fault tolerance by using map reduce functionality efficiently and effectively.

Proposed DPFPM algorithm has five steps. In first step it divide transaction database*DB*. In second step it constructs Global Frequent Header Table (*GFHTable*) using MapReduce tasks for first time. Third step generate conditional frequent patterns using map task second time. In fourth step each reducer combine all conditional frequent patterns of same key to generate frequent sets. In fifth step all reducers output is aggregated to generate final frequent sets.

Details of all steps are as follows –

A. Divide transaction database DB

Divide complete transaction database *DB* into number of available processors so that each individual processor works on the allocated *DB*in parallel. $DB = \{DB_i, DB_{i+1}, \ldots, DB_p\}$,where*p* is number of processors available to distribute the problem.

B. Construction of GFHTable

Using MapReduce programming model, the frequent support count of each item is calculated that appeared in the locally allottedDB_i. Then entire locally constructed header tables are merged to populate *GFHTable* on master node or at job node.

Algorithmic steps for construction of *GFHTable* are described below.

Step 1: Hadoop stack starts as many mappers as possibleto utilize available infrastructure at max.

Step 2: Complete *DB* is divided into number of Mappersand its transactions are read by Mapper function of Hadoop Stack

Step 3: Function GFHTMapper performs following steps taking transaction$T_i$from *DB* as input.

Step 3.1: foreach item from transaction T_i

Step 3.2.1: output write element and its frequency count 1

Step 4: Hadoop pass the Mapper output to number of reducers it creates. But number of reducers is depend on the number of groups (similar items/elements)

Step 5: Function GFHTReducer performs following steps taking input as elements grouped by element name

Step 5.1: long sum = 0;

Step 5.2: foreach element record

Step 5.2.1: sum = sum + element's frequency count

Step 5.3: outputs from each reducer started by Hadoop stack, element name and its global frequency count

Step6: Function aggregator prune infrequent items from *GFHTable* based on*minimum support threshold* ξInputto aggregator function is *minimum support threshold* ξ and output generated by GHTReducer Step 6.1: foreach element from GHTReducer

Step: 6.1.1: if (element.frequencyCount>= ξ)

Step 6.1.2: putInGFHTable(element);

Proposed methodology used for constructing GFHTable reduces communication overhead to calculate support of all items. It first calculates frequency count locally of local databases on each processor in parallel accelerating speed as described in step 1 to step 5. This reduces message exchange among processors. All local frequency counts are only then accumulated at master node to construct global header. As shown in step 6 all infrequent items are pruned from table and global frequent pattern header table (GFHTable) is constructed as final output. This step effectively uses map and reduce functionality to speed up the operation of mining.

C.Construction of Conditional Frequent Patterns

As the *GFHTable* is light weight in terms of memory, it is shared by all the nodes of the cluster. Each cluster node is responsible of rearranging the *DB* transaction pattern as in line with *GFHTable* and prunes the elements accordingly. As compared to standalone algorithms like FP-growth, proposed algorithm would not create global tree. Task of generation of frequent patterns is also distributed among processors and task of generating conditional patterns is done in parallel.

Here Mapper and Reducer perform different tasks than the traditional scatter and gather functionalities.

Here Mapper performs following steps:

i. It prunes the DB_i transaction to inline the transaction items as per the available *GFHTable*.

ii. According to the available *GFHTable*, it process the DB_i*'s* transaction and output the key-value pair, where key is conditional element for which conditional frequent pattern set is constructed as described in figure 1.

Algorithmic steps as explained above for FPM_Mapper for finding frequent patterns includes functions as described below. FPM-Mapper is main function where it calls generateConditionalPattern sub function. FPM-Mapper runs on all processors with multiple mappers running on individual processors. Conditional patterns are directly generated for each item

on each mapper on every processor as output of this function.

Function –FPM_Mapper to prune transactions and generate conditional frequent patterns

```
FunctionFPM_Mapper
input :GFHTable, Transaction T_i from DB
output: Conditional Frequent Patterns
# GFHTable is in local object to access it faster by current Mapper
Steps:
foreach transaction from DB
            Collect patternElements;
            foreach element of transaction T_i
            if (checkIfElementExistsInGFHTable(element))
            patternElements.add(element);                    }
endif
            #order pattern Elements using support frequency
            Prune-order(patternElements);
            #Generate Conditional Patterns
            generateConditionalPattern(patternElements);
            endfor
            endfor
```

Function – generate conditional frequent patterns

```
functiongenerateConditionalPattern
input :patternElements
output : conditional prefix-tree for each element
Steps: for each element from patternElements
construct conditional prefix-tree
HadoopContext.write(element, conditionalPrefix-tree);
endfor
```

FPM_Mapper first prune infrequent items from transactions from local database by checking with*GFHTable*. Then it calls generateConditionalPattern sub function which ultimately generates localconditional frequent patterns.

Figure1. shows architecture diagram of frequent pattern generation and does not include steps of *GFHTable* construction. Filesystem shown in figure is Hadoopfilesystem (HDFS) which is used for storing intermediate and final results of each mapper and reducer. HDFS provides reliability for storing huge datasets and stream intermediate results with high bandwidth to all nodes. The architecture of our system proves efficient use of MapReduce framework. Proposed architecture helps proposed algorithm to efficiently mine frequent patterns.

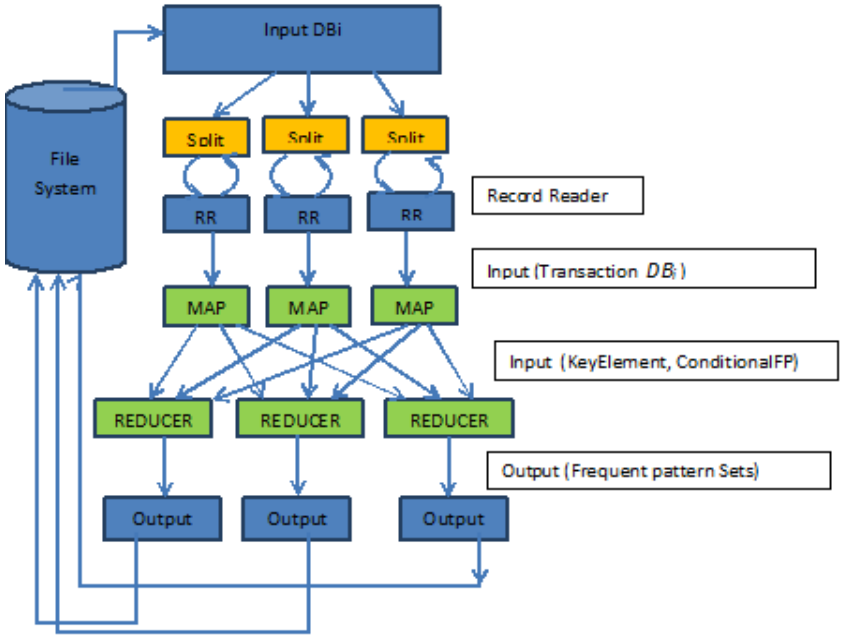

Figure1. Architecture diagram of DPFPM algorithm using MapReduce framework

D. Generation of Frequent Pattern Sets

When all mapper instances finished their work of construction of conditional frequent patterns, MapReduce framework helps to group all corresponding conditional frequent patterns transaction over available key element for which conditional frequent patterns are constructed.

Here Reducer performs following step:

At each reducer instance, it combines the different conditional frequent patterns over the same element key and generates the frequent pattern sets as an output.

```
FunctionFPM_Reducer
input : Conditional frequent patterns grouped by element Name
output: Frequent pattern Set
Steps:
foreachconditionalPattern from groupofConditionalPatterns
cfpHead = new ConditionalFP();
if (first ConditionalPattern)
cfpHead = contains conditionalPattern;
else
merge the already available pattern elementswith incoming matched
element patterns.
Increment the frequency counts base on the similar elements
update the conditional FPhead node
cfpHead.intersect(conditionalPattern);
```

```
endif
HadoopContext.write(groupName/*ElementName*/, cfpHead/*Data*/);
endfor
```

Each Reducer combines conditional patterns based on element key as shown in above algorithm. After combining all the patterns for each key, respective reducers do pruning of infrequent itemsets based on $DB = \{DB_i, DB_{i+1}, \ldots, DB_p\}$, Each reducer after pruning generates frequent sets for respective keys locally at each processor. This step reduces cost of Inter process communication as for each key frequent set are generated locally.

DPFPM algorithm is explained with following example.

Example: *Input Transaction Database* DBcontains 5 transactions as shown in table I. Each transaction contain m elements of I items.

M*inimum support threshold* $\xi$is considered for example.

TABLE I.TRANSACTION DATABASE DB

Transaction ID	Transactions
1	f, a, c, d, g,i, m, p
2	a, b, c, f , l, m, o
3	b, f, h, j, o
4	b, c, k, s, p
5	a, f, c, e, l, p, m, n

GFHTable generated using GFHTMapper and GFHTReducer functions is as shown in Table II.

TABLE II.*GFHTABLE*

Item Identifier	c	f	a	b	m	p
Support count	4	4	3	3	3	3

After construction of *GFHTable*using Map Reduce framework for first, second step is carried out using FPM-Mapper and FPM-Reducer functions, where it uses MapReduce framework for second time.

Example –FPM-Mapper functions in following way:

Transactions are mapped to mapper where sorting and pruning of transactions based on *GFHTable* takes place locally on each processor. Each mapper generates as output (Key: Conditional Pattern) pair.

TABLE III. FPM-MAPPER OUTPUTS

Mapperinput (db transaction pattern input)	Sorted and pruned transaction patterns	Mapper output – conditional frequentpatterns (Key Element: Conditional Pattern)
f, a,c,d,g,i,m,p	c,f,a,m,p	p: c f a m m: c f a a: c f f: c c: None
a,b,c,f,l,m,o	c,f,a,b,m	m: c f a b b: c f a a: c f f: c c: None
b,f,h,j,o	f,b	b: f f: None
b, c,j,s,p	c,b,p	p: c b b: c c: None
a,f,c,e,l,p,m,n	c,f,a,m,p	p: c f a m m: c f a a: c f f: c c: None

As shown in table III conditional patterns of each item are created as output to mapper function locally on each processor.

Example –FPM-Reducer functions in following way:

Each Reducer then combines conditional patterns based on element key. After combining all the patterns for eachkey, respective reducers do pruning of infrequent Itemsets based based on*minimum support threshold* $\xi = 3$. Each reducer after pruning generates frequent sets for respective keys locally at each processor. This step reduces cost of Inter process communication as for each key frequent set are generated locally.

As shown in table IV output of FPM-Mapper is passed to reducer where for example first reducer collects all patterns based on element 'a' that is "a: {c, f} | {c, f} | {c, f}". Then it prunes infrequent items from the set based on $\xi = 3$.For first reducer "a:{c, f}" is frequent so no pruning is done. Finallyconditional patterns of each item are created as output to mapper functionlocally on each processor.

TABLE IV. FPM-REDUCER OUTPUTS

Reducer combines conditional Patterns Base on Element Key	Pruned infrequent items based on ξ	Final Frequent Sets
a: {c, f} | {c, f} | {c, f}	a:{c, f}	{a}, {a, f}, {a, f, c}, {a, c}
m: {c, f, a} | {c, f, a, b}| {c, f, a}	m: {a, f , c}	{m}, {m, a}, {m, a, f}, {m, a, f, c}, {m, f}, {m, a, c},{m, f, c}, {m, c}
p: {c, f, a, m} | {c, b} | {c, f, a, m}	p: {c}	{p}, {p, c}
b: {f} | {c} | {c, f, a}	b: {}	{b}
c: {}	c: {}	{c}
f: {c} | {c}| {c}	f: {c}	{f} {f, c}

E. AGGREGATION

The results of the above step of reducer are aggregated as final outcome of this algorithm to generate frequent patterns available in complete *DB*passed as input to the proposed DPFPM algorithm. Function aggregator takes output of all reducers and generates final frequent patterns. This step is carried out on master node.

```
function aggregator
input : fetch all the reducers output
output: generate aggregate output as frequent pattern sets
#Scan through the entire Reducer of Hadoop stack
#Read all the part-r-success-0000* files
cfpHeadList = null;
foreach file from foundFiles
if (cfpHeadList == null)
#read object from file and align here
cfpHeadList = fetchFrequentPatternSets(file);
else
#concatenate objects from file
cfpHeadList.addFrequentPatternSets(file);
endif
endfor
foreachcfpNode from cfpHeadList
#Print element and its frequent sets
System.out.println(elementName, freqquentSets);
endfor
```

Proposed DPFPM algorithm uses MapReduce framework effectively to achieve performance goals in terms of execution efficiency and scalability.

V. EXPERIMENTALRESULTS

For Proposed DPFPM algorithm experiments are performed on Hadoop Cluster of 5 Nodes (1 master, 4 slaves). Each node has Core2Duo processors (running at 2.60GHz) and 2GB memory. Several real and synthetic datasets are used to test the performance of the algorithm.

Experiments are performed with relative minimum support threshold range from 2 to 10. To test scalability experiments are performed on nodes scale from 1 to 5. To prove the performance results are compared with DH-Trie algorithm [10].

Comparison with other algorithms-

TABLE V. COMPARISON OF ALGORITHMS

Algorithm	Number of Transactions	Frequent Pattern generation time
DH-Trie	1,00,00,000	1062.26 seconds
DPFPM	3,16,80,064	718.24 seconds

Comparison of proposed DPFPM algorithm with DH-Trie algorithm as shown in Table V proves that DPFPM algorithm is order of magnitude faster than DH-Trie algorithm. Proposed algorithm also proves very good scalability when tested on the scale of 1 node to 5 node cluster. Figure 2 shows experimental results for dataset Kosarak1G.dat which contain 3,16,80,064 number of transactions. Experiments are performed for minimum support threshold of 4%. Figure 2 graph shows number of nodes /processors on X-axis and *GFHTable* and Frequent pattern generation time in seconds on Y-axis.

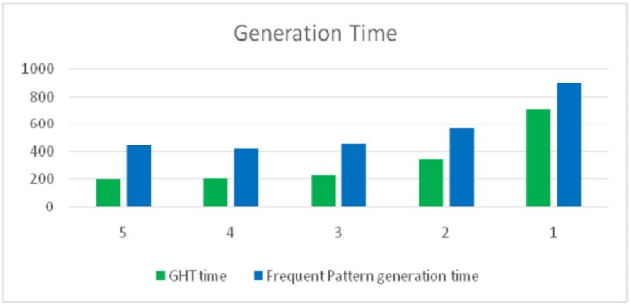

Figure2. Graph – Number of Nodes vs. Time

TABLE VI. READINGS FOR VARIOUS MINIMUM SUPPORTS

Minimum Support	Minimum count	GFHTable construction time	Frequent Pattern generation time
2	633601	193.02 seconds	525.22 seconds
4	1267202	198.03 seconds	444.06 seconds
5	1584003	192.1 seconds	536.63 seconds
6	1900803	190.97 seconds	534.27 seconds
8	2534404	190.99 seconds	486.16 seconds
10	3168006	189.32 seconds	420.32 seconds

TABLE VI shows readings forKosarak1G.dat. Graph of these readings are plotted in Figure 3.

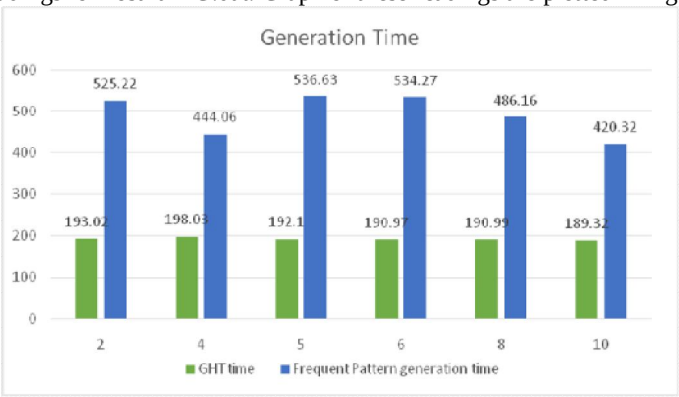

Figure 3. Graph – Frequent pattern generation time for various support thresholds

Dataset – Kosarak1G.dat
Graph – Minimum Support vs. Time
Number of Transactions – 3,16,80,064

Figure 3 graph shows minimum support count on X-axis and GFHTable and Frequent pattern generation time in seconds on Y-axis.
Experimental results prove efficiency of proposed DPFPM algorithm. It also proves good scalability withrespect to various minimum supports.

VI. CONCLUSION

Proposed DPFPM algorithm shows best performance results for large databases using HadoopMapReduce framework. Proposed algorithm implement parallel processing for header table generation and mining frequent pattern both, speeding up the execution. Proposed algorithms partition/distribute the database in such a way that, it works independently and parallel at each local node reducing the communication overhead in both the stages. It would not construct global tree but local conditional patterns are given to reducer and itperforms task of frequent pattern generation, reducing communication cost, and speed up mining process. Experimental result shows that parallel and distributed algorithm efficiently handles the scalability for very large databases

REFERENCES

[1] R.Agrawal, T.Imielinski and A.N.Swami, "Mining association rules between set of items in large databases", in Proceedings of the ACM SIGMOD Conference on management of data, 1993 pp. 207–216.
[2] R. Agrawal and J.C.Shafer, "Parallel Mining of Association Rules: Design, Implementation and experience," Technical Report TJ10004, IBM Research Division, Almaden Research Center, 1996.
[3] M. J.Zaki, "Parallel and Distributed AssociationMining:A survey," *IEEE Concurrency*, vol. 7(4), 1999, pp. 14-25.
[4] Y. Ye, C. Chiang, "A Parallel Apriori Algorithm for Frequent Itemsets Mining, "Fourth International Conference on Software Engineering Research, Management and Applications, 2006, pp. 87 – 94.
[5] K .M. Yu, J. Zhou,T. P. Hong and J. L. Zhou, "A load-balanced distributed parallel mining algorithm",Elsevier-Expert Systems with Applications37 (2010) pp. 2459–2464.
[6] XinYue Yang, Zhen Liu and Yan Fu,"MapReduce as a Programming Model for Association Rules Algorithm on Hadoop", in: Information Sciences and Interaction Sciences (ICIS), 2010.
[7] J. Han, J. Pei and Y.Yin, "Mining frequent patterns without candidate generation", In Proceedings of the ACM-SIGMOD conference management of data, 2000, pp. 1–12.
[8] Syed KhairuzzamanTanbeer, ChowdhuryFarhan Ahmed and Byeong-SooJeong, "Parallel and Distributed Frequent Pattern Mining in Large Databases",11th IEEE International Conference on HPCC,2009.
[9] Kun-Ming Yu, Jiayi Zhou, "Parallel TID-based frequent pattern mining algorithm on a PC Cluster and grid computing system", Expert Systems with Applications 37 pp. 2486–2494, 2010.
[10] Lai Yang, Zhongzhi Shi, Li XU, Fan Liang and IlanKirsh "DH-TRIE Frequent Pattern Mining on Hadoop using JPA", IEEE International Conference on Granular Computing ,2011.
[11] Apache Hadoop. http://hadoop.apache.org/
[12] K. Shvachko, HairongKuang, Sanjay Radia, Robert Chansler, "The Hadoop Distributed File System",In Proceedings of the IEEE 26th Symposium on Mass Storage Systems and Technologies, 2010.
[13] J. Dean, S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," In Proc. of the 6th symposium on Operating Systems Design and Implementation, San Francisco CA, Dec. 2004.
[14] S. Parthasarathy, M. J. Zaki, M. Ogihara and W. Li., "Parallel Data Mining for Association Rules on Shared-Memory Systems", Knowledge and Information Systems, 3(1):1–29,February 2001.
[15] R.C. Agrawal, C.C. Agrawal, and V.V.V. Prasad, "A tree projection Algorithm for generation of frequent itemsets," *Journal of Parallel and Distributed Computing*, Vol.61, No. 3, pp: 350–371,March 2001.
[16] J. Han, Cheng Hong, Xin Dong, Yan and Xifeng, "Frequent pattern mining: current status and future direction," in *Data Mining and Knowledge Discovery*, Vol. 15, No1, pp. 55-86 , August 2007.
[17] Kawuu W. Lin and Yu-Chin Lo,"Efficient algorithms for frequent pattern mining in many-task computing environments", Knowledge-Based Systems,2013.